SCALABLE MULTI-OUTPUT LABEL PREDICTION: FROM CLASSIFIER CHAINS TO CLASSIFIER TRELLISES

J. Read[1]*, L. Martino*[2]*, P. M. Olmos*[3]*, David Luengo*[4]

[1] Dep. of Computer Science, Aalto University and HIIT, Helsinki, Finland (jesse.read@aalto.fi).
[2] Dep. of Mathematics and Statistics, University of Helsinki, Helsinki (Finland).
[3] Dep. of Signal Theory and Communications, Universidad Carlos III de Madrid (Spain) .
[4] Dep. of Circuits and Systems Engineering, Universidad Politécnica de Madrid, (Spain).

ABSTRACT

Multi-output inference tasks, such as multi-label classification, have become increasingly important in recent years. A popular method for multi-label classification is *classifier chains*, in which the predictions of individual classifiers are cascaded along a chain, thus taking into account inter-label dependencies and improving the overall performance. Several varieties of classifier chain methods have been introduced, and many of them perform very competitively across a wide range of benchmark datasets. However, scalability limitations become apparent on larger datasets when modeling a fully-cascaded chain. In particular, the methods' strategies for discovering and modeling a good chain structure constitutes a mayor computational bottleneck. In this paper, we present the *classifier trellis* (CT) method for scalable multi-label classification. We compare CT with several recently proposed classifier chain methods to show that it occupies an important niche: it is highly competitive on standard multi-label problems, yet it can also scale up to thousands or even tens of thousands of labels.

Keywords: classifier chains; multi-label classification; multi-output prediction; structured inference; Bayesian networks

1. INTRODUCTION

Multi-output classification (MOC) (also known variously as multi-target, multi-objective, and multidimensional classification) is the supervised learning problem where an instance is associated with a set of qualitative discrete variables (a.k.a. *labels*), rather than with a single variable[1]. Since these label variables are often strongly correlated, modeling the dependencies between them allows MOC methods to improve their performance at the expense of an increased computational cost. Multi-label classification (MLC) is a special case of MOC where all the labels are binary; it has already attracted a great deal of interest and development in machine learning literature over the last few years. In [27], the authors give a recent review of, and many references to, a number of recent and popular methods for MLC. Figure 1 shows the relationship between different classification paradigms, according to the number of labels ($L = 1$ vs. $L > 1$) and their type (binary $[K = 2]$ or not $[K > 2]$).

There are a vast range of active applications of MLC, including tagging images, categorizing documents, and labelling video and other media, and learning the relationship among genes and biological functions. Labels (e.g., tags, categories, genres) are either relevant or not. For example, an image may be labelled `beach` and `urban`; a news article may be sectioned under `europe` and `economy`. Relevance is usually indicated by 1, and irrelevance by 0. The general MOC scheme may add other information such as month, age, or gender. Note that month $\in \{1, \ldots, 12\}$ and therefore is not simply irrelevant or not. This MOC task has received relatively less attention than MLC (although there is some work emerging, e.g., [25] and [17]). However, most MLC-transformation methods (e.g., treating each label variable as a separate multi-class problem) are equally applicable to MOC. Indeed, in this paper we deal with a family of methods based on this approach. Note also that, as any integer can be represented in binary form (e.g., $3 \Leftrightarrow [0, 0, 0, 1, 1]$), any MOC task can 'decode' into a MLC task and vice versa.

In this paper, we focus on *scalable* MLC methods, able to effectively deal with large datasets at feasible complexity. Many recent MLC methods, particularly those based on classifier chains, tend to be over engineered, investing evermore computational power to model label dependencies, but presenting poor scalability properties. In the first part of the paper, we review some

[1] We henceforth try to avoid the use of the term 'class'; it generates confusion since it is used variously in the literature to refer to both the target variable, and a value that the variable takes. Rather, we refer to *label* variables, each of which takes a number of *values*.

state-of-the-art methods from the MLC and MOC literature to show that the more powerful solutions are not well-suited to deal with large-size label sets. For instance, *classifier chains* [3, 20] consider a full cascade of labels along a chain to model their joint probability distribution and, either they explore all possible label orders in the chain, incurring in exponential complexity with the number of labels, or they compare a small subset of them chosen at random, which is ineffective for large dimensional problems.

The main contribution of the paper is a novel highly-scalable method: the *classifier trellis* (CT). Rather than imposing a long-range and ultimately computationally complex dependency model, as in classifier chains, CT captures the essential dependencies among labels very efficiently. This is achieved by considering a predefined trellis structure for the underlying graphical model, where dependent nodes (labels) are sequentially placed in the structure according to easily-computable probabilistic measures. Experimental results across a wide set of datasets show that CT is able to scale up to large sets (namely, thousands and tens of thousands of labels) while remaining very competitive on standard MLC problems. In fact, in most of our experiments, CT was very close to the running time of the naive baseline method, which neglects any statistical dependency between labels. Also, an ensemble version of CT, where the method is run multiple times with different random seeds and classification is done through majority voting, does not significantly outperform the single-shot CT. This demonstrates that our method is quite robust against initialization.

The paper is organized as follows. First, in Section 2 we formalize the notation and describe the problem's setting. In Section 3 we review some state-of-the-art methods from the MLC and MOC literature, as well as their various strategies for modeling label dependence. This review is augmented with empirical results. In Section 4 we make use of the studies and theory from earlier sections to present the classifier trellis (CT) method. In Section 5 we carry out two sets of experiments: firstly we compare CT to some state-of-the-art multi-label methods on an MLC task; and secondly, we show that CT can also provide competitive performance on typical structured output prediction task (namely localization via segmentation). Finally, in Section 6 we discuss the results and take conclusions.

Two appendixes have been included to help the readability of the paper and support the presented results. In A, we compare two low-complexity methods to infer the label dependencies from training data. In B we review Monte Carlo methods, which are required in this paper to perform probabilistic approximate inference of the label set associated to a new test input.

2. PROBLEM SETUP AND NOTATION

Following a standard machine learning notation, the n-th feature vector can be represented as

$$\mathbf{x}^{(n)} = [x_1^{(n)}, \dots, x_D^{(n)}]^\top \in \boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_D \subseteq \mathbb{R}^D,$$

where D is the number of features and $\mathcal{X}_d$ ($d = 1, \dots, D$) indicates the support of each feature. In the traditional *multi-class classification* task, we have a single target variable which can take *one* out of K values, i.e.,

$$y^{(n)} \in \mathcal{Y} = \{1, \dots, K\},$$

and for some test instance $\mathbf{x}^*$ we wish to predict

$$\hat{y} = h(\mathbf{x}^*) = \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|\mathbf{x}^*), \tag{1}$$

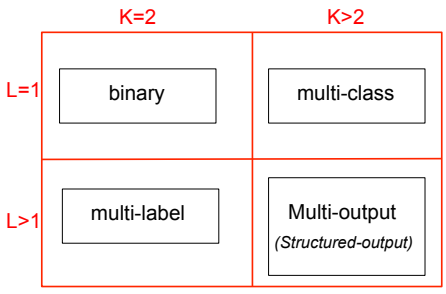


Fig. 1: Different classification paradigms: L is the number of *labels* and K is the number of *values* that each label variable can take.

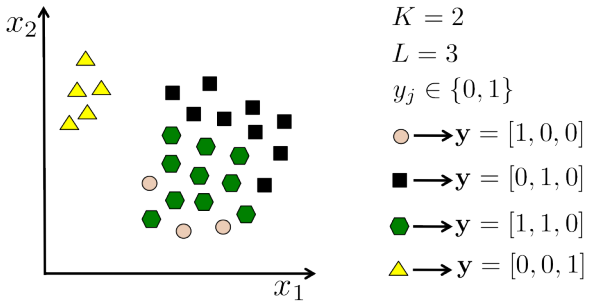


Fig. 2: Toy example of multi-label classification (MLC), with $K = 2$ possible values for each label and $L = 3$ labels (thus $y_j \in \{0, 1\}$ for $j = 1, 2, 3$) and, implicitly, $D = 2$ features. Circles, squares and triangles are elements with only one active label (i.e., either $y_1 = 1$ or $y_2 = 1$, but not both). Hexagons show vectors such that $y_1 = y_2 = 1$.

in such a way that it coincides with the true (unknown) test label with a high probability[2]. Furthermore, the conditional distribution $p(y|\mathbf{x})$ is usually unknown and has to be estimated during the classifier construction stage. In the standard setting, classification is a supervised task where we have to infer the model h from a set of N labelled examples (training data) $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$, and then apply it to predict the labels for a set of novel unlabelled examples (test data). This prediction phase is usually straightforward in the single-output case, since only one of K values needs to be selected.

In the *multi-output classification* (MOC) task, we have L such output labels,

$$\mathbf{y}^{(n)} = [y_1^{(n)}, \ldots, y_L^{(n)}]$$

where

$$y_\ell^{(n)} \in \mathcal{Y}_\ell = \{1, \ldots, K_\ell\},$$

with $K_\ell \in \mathbb{N}_+$ being the finite number of values associated with the ℓ-th label. For some test instance $\mathbf{x}^*$, and provided that we know the conditional distribution $p(\mathbf{y}|\mathbf{x})$, the MOC solution is given by

$$\hat{\mathbf{y}} = h(\mathbf{x}^*) = \underset{\mathbf{y} \in \boldsymbol{\mathcal{Y}}}{\operatorname{argmax}}\, p(\mathbf{y}|\mathbf{x}^*). \tag{2}$$

Once more, $p(\mathbf{y}|\mathbf{x})$ is usually unknown and has to be estimated from the training data, $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{N}$, in order to construct the model h. Therein precisely lies the main challenge behind MOC, since h must select one out of $|\mathcal{Y}| = K^L$ possible values[3]; clearly a much more difficult task than in Eq. (1). Furthermore, finding $\hat{\mathbf{y}}$ for a given $\mathbf{x}^*$ and $p(\mathbf{y}|\mathbf{x})$ is quite challenging from a computational point of view for large values of K and L [17, 25].

In MLC, all labels are binary labels, namely $K_\ell = 2$ for $\ell = 1, \ldots, L$, with the two possible label values typically notated as $y_\ell \in \{0, 1\}$ or $y_\ell \in \{-1, +1\}$. Figure 2 shows one toy example of MLC with three labels (thus $\mathbf{y} \in \{0, 1\}^3$). Because of the strong co-occurrence, we can interpret that the first label ($y_1 = 1$) implies the second label ($y_2 = 1$) with high probability, but not the other way around. When learning the model $h(\mathbf{x})$ in (2), the goal of MLC (and MOC in general) is capturing this kind of dependence among labels in order to improve classification performance; and to do this efficiently enough to scale up to the size of the data in the application domains of interest. This typically means connecting labels (i.e., learning labels together) in an appropriate structure. Table 1 summarizes the main notation used in the paper.

3. BACKGROUND AND RELATED WORK

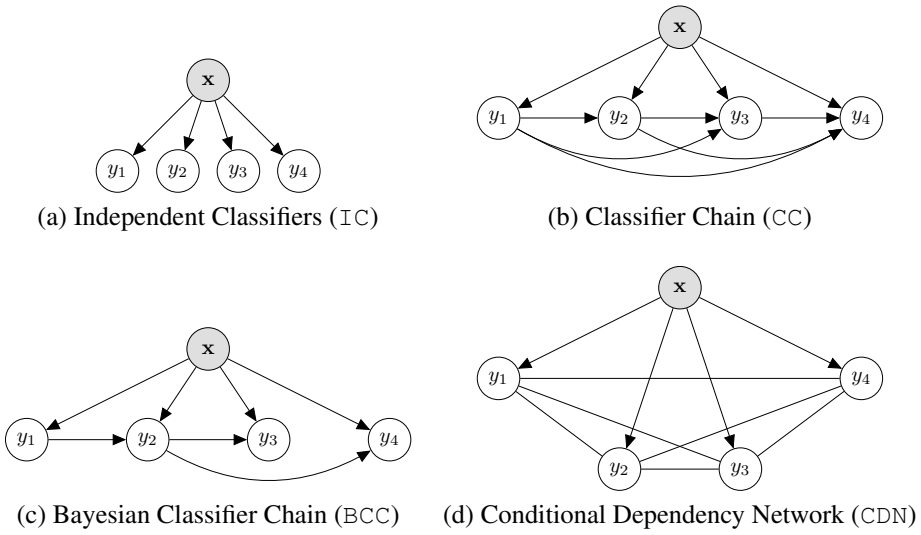
In this section, we step through some of the most relevant methods for MLC/MOC recently developed, as well as several works specifically related to the novel method, presented in later sections. All the methods discussed here, and also the CT method presented in Section 4, aim to build a model for $h(\mathbf{x})$ in (2) by first selecting a suitable model for the label joint posterior distribution $p(\mathbf{y}|\mathbf{x})$ and then using this model to provide a prediction $\hat{\mathbf{y}}$ to a new test input $\mathbf{x}^*$. It is in the first step where state-of-the-art methods present a complexity bottleneck to deal with large sets of labels and where CT offers a significantly better complexity-performance trade-off.

[2]Eq. (1) corresponds to the widely used maximum a posteriori (MAP) estimator of y^* given $\mathbf{x}^*$, but other approaches are possible.

[3]A simplification of $K_1 \times K_2 \times \cdots \times K_L$ (to keep notation cleaner).

Table 1: Summary of the main notation.

Notation	Description
$\mathbf{x} = [x_1, \ldots, x_D]^\top$	instance / input vector; $\mathbf{x} \in \mathbb{R}^D$
$y \in \{0, 1\}$	a label (binary variable)
$y \in \{1, \ldots, K\}$	an output (multi-class variable), K possible values
$\mathbf{y} = [y_1, \ldots, y_L]^\top$	L-dimensional label/output vector
$\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$	Training data set, $n = 1, \ldots, N$
$\hat{y} = h(\mathbf{x}^*)$	binary or multi-class classification (test instance $\mathbf{x}^*$)
$\hat{\mathbf{y}} = h(\mathbf{x}^*)$	multi-label multi-output classification (MLC, MOC)



(a) Independent Classifiers (IC) (b) Classifier Chain (CC)

(c) Bayesian Classifier Chain (BCC) (d) Conditional Dependency Network (CDN)

Fig. 3: Several multi-label methods depicted as directed/undirected graphical models.

3.1. Independent Classifiers

A naive solution to multi-output learning is training L K-class models as in Eq. (1), i.e., L *independent classifiers* (IC),[4] and using them to classify L times a test instance $\mathbf{x}^*$, as $[\hat{y}_1, \ldots, \hat{y}_L] = [h_1(\mathbf{x}^*), \ldots, h_L(\mathbf{x}^*)]$. IC is represented by the directed graphical model shown in Figure 3 (a). Note that this approach implicitly assumes the independence among the target variables, i.e., $p(\mathbf{y}|\mathbf{x}) \equiv \prod_{\ell=1}^{L} p(y_\ell|\mathbf{x})$, which is not the case in most (if not all) multi-output datasets.

3.2. Classifier Chains

The *classifier chains* methodology is based on the decomposition of the conditional probability of the label vector $\mathbf{y}$ using the product rule of probability:

$$p(\mathbf{y}|\mathbf{x}) = p(y_1|\mathbf{x}) \prod_{\ell=2}^{L} p(y_\ell|y_1, \ldots, y_{\ell-1}, \mathbf{x}) \tag{3}$$

$$\approx f_1(\mathbf{x}) \prod_{\ell=2}^{L} f_\ell(\mathbf{x}, y_1, \ldots, y_{\ell-1}), \tag{4}$$

which is approximated with L probabilistic classifiers, $f_\ell(\mathbf{x}, y_1, \ldots, y_{\ell-1})$. As a graphical model, this approach is illustrated by Figure 3 (b).

[4] In the MLC literature, the IC approach is also known as the *binary relevance* method.

The complexity associated to learn Eq. (4) increases with L, but with a fast greedy inference, as in [20], it reduces to

$$\hat{y}_\ell = \underset{y_\ell}{\operatorname{argmax}}\, p(y_\ell | y_1, \ldots, y_{\ell-1}, \mathbf{x}), \tag{5}$$

for $\ell = 1, \ldots, L$. This is not significant for most datasets, and time complexity is close to that of IC in practice. In fact, it would be identical if not for the extra $y_1, \ldots, y_\ell - 1$ attributes.

With greedy inference comes the concern of error propagation along the chain, since an incorrect estimate $\hat{y}_\ell$ will negatively affect all following labels. However, this problem is not always serious, and easily overcome with an ensemble [20, 3]. Therefore, although there exist a number of approaches for avoiding error propagation via exhaustive iteration or various search options [3, 13, 17], we opt for the ensemble approach.

3.3. Bayesian Classifier Chains

Instead of considering a fully parameterized Markov chain model for $p(\mathbf{y}|\mathbf{x})$, we can use a simpler Bayesian network. Hence, (3) becomes

$$p(\mathbf{y}|\mathbf{x}) = \prod_{\ell=1}^{L} p(y_\ell | \mathbf{y}_{\mathsf{pa}(\ell)}, \mathbf{x}), \tag{6}$$

where $\mathsf{pa}(\ell)$ are the parents of the ℓ-th label, as proposed in [25, 26], known as as Bayesian Classifier Chains (BCC), since it may remind us of Bayesian networks. Using a structure makes training the individual classifiers faster, since there are fewer inputs to them, and also speeds up any kind of inference. Figure 3 (c) shows one example of many possible such network structures.

Unfortunately, finding the optimal structure is NP hard due to an impossibly large search space. Consequently, a recent point of interest has been finding a good suboptimal structure, such that Eq. (6) can be used. The literature has focused around the idea of label dependence (see [5] for an excellent discussion). The least complex approach is to measure *marginal label dependence*, i.e., the relative co-occurrence frequencies of the labels. Such approach has been considered in [25, 8]. In the latter, the authors exploited the *frequent sets* approach [1], which measures the co-occurrence of several labels, to incorporate edges into the Bayesian network. However, they noted problems with attributes and negative co-occurrence (i.e., mutual exclusiveness; labels whose presence indicate the absence of others). The resulting algorithm (hereafter referred to as the FS algorithm) can deal with moderately large datasets, but the final network construction approach ends up being rather involved.

Finding a graph based on *conditional* label dependence is inherently more demanding, because the input feature space must be taken into account, i.e., classifiers must be trained. Of course, training time is a strongly limiting factor here. However, a particularly interesting approach to modelling conditional dependence, the so-called LEAD method, was presented in [26]. This scheme tries to remove first the dependency of the labels on the feature set, which is the common parent of all the labels, to facilitate learning the label dependencies. In order to do so, LEAD trains first an independent classifier for each label (i.e., it builds L independent models, as in the IC approach), and then uses the dependency relations in the residual errors of these classifiers to learn a Bayesian network following some standard approach (the errors can in fact be treated exactly as if they were labels and plugged, e.g., into the FS approach). LEAD is thus a fast method for finding conditional label dependencies, and has shown good performance on small-sized datasets.

Neither the FS nor the LEAD methods assume any particular constraint on the underlying graph and are well suited for MOC in the high dimensional regime because of their low complexity. However, if the underlying directed graph is sparse, the PC algorithm and its modifications [12, 24] are the state-of-the-art solution in directed structured learning. The PC-algorithm runs in the worst case in exponential time (as a function of the number of nodes), but if the true underlying graph is sparse, this reduces to a polynomial runtime. However, this is typically not the case in MLC/MOC problems.

For the sake of comparison between the different MLC/MOC approaches, in this paper we only consider the FS and LEAD methods to infer direct dependencies between labels. In order to delve deeper into the issue of structure learning using the FS and LEAD methods, in A we have generated a synthetic dataset, where the underlying structure is known, and compare their solutions and the degree of similarity with respect to the true graphical model. As these experiments illustrate, one of the main problems behind learning the graphical model structure from scratch is that we typically get too dense networks, where we cannot control the complexity associated to training and evaluating each one of the probabilistic classifiers corresponding to the resulting factorization in (6). This issue is solved by the classifier trellis method proposed in Section 4.

3.4. Conditional Dependency Networks (CDN)

Conditional Dependency Networks represent an alternative approach, in which the conditional distribution $p(\mathbf{y}|\mathbf{x})$ factorizes according to an undirected graphical model, i.e.,

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_{q=1}^{Q}\phi_q(\mathbf{y}_q|\mathbf{x}), \tag{7}$$

where Z is a normalizing constant, $\phi_I(\cdot)$ is a positive function or potential, and $\mathbf{y}_q$ is a subset of the labels (a *clique* in the undirected graph). The notion of directionality is dropped, thus simplifying the task of learning the graph structure. Undirected graphical models are more natural for domains such as spatial or relational data. Therefore, they are well suited for tasks such as image segmentation (e.g., [4], [14]), and regular MLC problems (e.g., [7]).

Unlike classifier chain methods, a CDN does not construct an approximation to $p(\mathbf{y}|\mathbf{x})$ based on a product of probabilistic classifiers. In contrast, for each conditional probability of the form

$$p(y_\ell|y_1^{(t)},\ldots,y_{\ell-1}^{(t)},y_{\ell+1}^{(t-1)},\ldots,y_L^{(t-1)},\mathbf{x}), \tag{8}$$

a probabilistic classifier is learnt. In an undirected graph, where all the labels that belong to the same clique are connected to each other, it is easy to check that

$$p(y_\ell|y_1^{(t)},\ldots,y_{\ell-1}^{(t)},y_{\ell+1}^{(t-1)},\ldots,y_L^{(t-1)},\mathbf{x}) = p(y_\ell|\mathbf{y}_{\mathsf{ne}(\ell)},\mathbf{x}), \tag{9}$$

where $\mathbf{y}_{\mathsf{ne}(\ell)}$ is the set of variables connected to y_ℓ in the undirected graph[5]. Finally, $p(y_\ell|\mathbf{y}_{\mathsf{ne}(\ell)},\mathbf{x})$ for $\ell = 1,\ldots,L$ is approximated by a probabilistic classifier $f_\ell(\mathbf{y}_{\mathsf{ne}(\ell)},\mathbf{x})$.

In order to classify a new test input $\mathbf{x}^*$, approximate inference using Gibbs sampling is a viable option. In B, we present the formulation of Monte Carlo approaches (including Gibbs sampling) specially tailored to perform approximate inference in MLC/MOC methods based on Bayesian networks and undirected graphical models.

3.5. Other MLC/MOC Approaches

A final note on related work: there are many other 'families' of methods designed for multi-label, multi-output and structured output prediction and classification, including many 'algorithm adapted' methods. A fairly complete and recent overview can be seen in [27] for example. However, most of these methods suffer from similar challenges as the classifier chains family; and similarly attempt to model dependence yet remain tractable by using approximations and some form of randomness [22, 18, 23]. To cite a more recent example, [21] uses 'random graphs' in a way that resembles [9]'s CDN, since it uses undirected graphical models, and [25]'s BCC in the sense of the randomness of the graphs considered.

3.6. Comparison of State-of-the-art Methods for Extracting Structure

It is our view that many methods employed for multi-label chain classifiers have not been properly compared in the literature, particularly with regard to their method for finding structure. There is not yet any conclusive evidence that modelling the marginal dependencies (among Y) is enough, or whether it is advisable to model the conditional dependencies also (for best performance); and how much return one gets on a heavy investment in searching for a 'good' graph structure, over random structures.

We performed two experiments to get an idea; comparing the following methods:

IC	Labels are independent, no structure.
ECC	Ensemble of 10 CC, each with random order [20]
EBCC-FS	Ensemble of 10 BCCs [25], based on marginal dependence [8]
EBCC-LEAD	as above, but on conditional dependence, as per [26]
OCC	the *optimal* CC – of all $L!$ possible (complete) chain orders

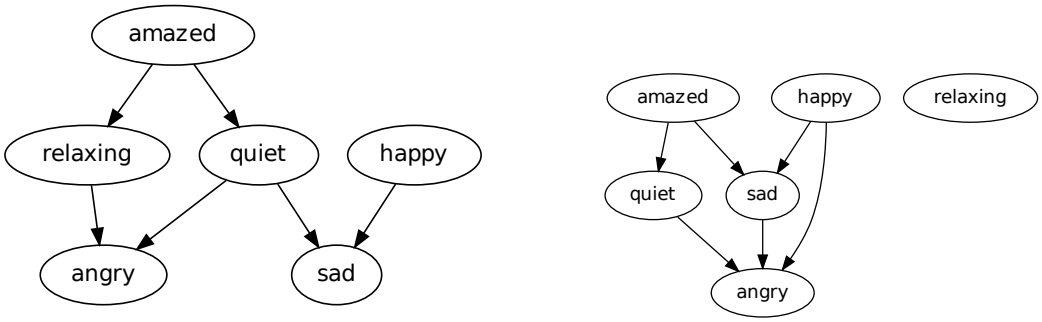
Results are shown in Table 2 of 5×CV (Cross validation) on two small real-world datasets (Music and Scene); small to ensure a comparison with OCC ($L! = 720$ possible chain orderings).

[5] In other words, $\mathbf{y}_{\mathsf{ne}(\ell)}$ is the so called *Markov blanket* of y_ℓ [2].

Table 2: Comparison of the classification accuracy of existing methods with $5\times$ CV (Cross validation). We chose the two smallest datasets so that the 'optimal' OCC could complete.

	IC	ECC	OCC	EBCC-FS	EBCC-LEAD
Music	0.517 ± 0.03	0.588 ± 0.03	0.580 ± 0.04	0.582 ± 0.02	0.578 ± 0.03
Scene	0.595 ± 0.02	0.698 ± 0.01	0.699 ± 0.01	0.644 ± 0.02	0.645 ± 0.03

Figure 4 shows an example of the structure found by BCC-FS and BCC-LEAD in a real dataset, namely Music (emotions associated with pieces of music). As a base classifier, we use support vector machines (SVMs), fitted with logistic models (as according to [11]) in order to obtain a probabilistic output, with the default hyper-parameters provided in the SMO implementation of the Weka framework [10]. Table 2 confirms that IC's assumption of independence harms its performance, and that the bulk of the MOC literature is justified in trying to overcome this. However, it also suggests that investing factorial and exponential time to find the best-fitting chain order and label combination (respectively) does not guarantee the best results. In fact, by comparing the results of ECC with EBCC-LEAD and EBCC-FS, even the relatively higher investment in conditional label dependence over marginal dependence (EBCC-LEAD vs EBCC-FS) does not necessarily pay off. Finally, ECC's performance is quite close to MCC. Surprisingly, the ECC method tends to provide excellent performance, even though it only learns randomly ordered (albeit fully connected) chains. However, as discussed in Section 3.2, its complexity is prohibitively large in high dimensional MLC problems.



(a) (b)

Fig. 4: Graphs derived from the Music dataset, with links based on (a) marginal dependence (FS, label-frequency) and (b) conditional dependence (LEAD, error-frequency). Here we have based the links on mutual information, therefore links represent both co-occurrences (e.g., quiet $\rightarrow$ sad) and mutual exclusiveness (e.g., happy $\rightarrow$ sad). Generally, we see that the graph makes intuitive sense; amazed and happy are neither strongly similar nor opposite emotions, and thus there is not much benefit in modelling them together; same with angry and sad.

4. A SCALABLE APPROACH: CLASSIFIER TRELLIS (CT)

The goal for a highly scalable CC-based method brings up the common question: which structure to use. On the one hand, ignoring important dependency relations will harm performance on typical MLC problems. On the other hand, assumptions must be made to scale up to large scale problems. Even though in certain types of MLC problems there is a clear notion of the local structure underlying the labels (e.g., in image segmentation pixels next to each other should exhibit interdependence), this assumption is not valid for general MLC problems, where the 4th and 27th labels might be highly correlated for example. Therefore, we cannot escape the need to discover structure, but we must do it efficiently. Furthermore, the structure used should allow for fast inference.

Our proposed solution is the *classifier trellis* (CT). To relieve the burden of specification 'from scratch', we maintain a fixed structure, namely, a *lattice* or *trellis* (hence the name). This escapes the high complexity of a complete structure learning (as in

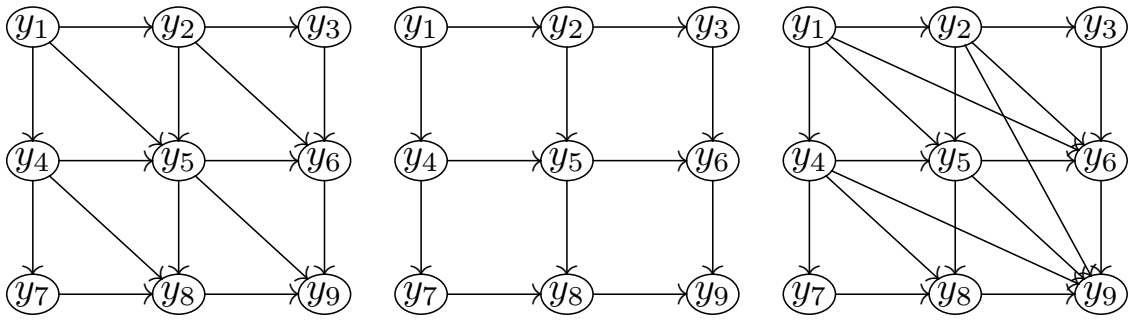


Fig. 5: Three possible directed trellises for $L = 9$. Each trellis is defined by a fixed pattern for the parents of each vertex (varying only near the edges, where parents are not possible). Note that no directed loops can exist.

[20] and [3]), and at the same time avoids the complexity involved in discovering a structure (e.g., [25] and [26]). Instead, we a impose a structure a-priori, and only seek an improvement to the order of labels within that structure.

Figure 5 gives three simple example trellises for $L = 9$ (specifically, we use the first one in experiments). Each of the vertices $\ell \in \{1, \ldots, L\}$ of the trellis corresponds to one of the labels of the dataset. Note the relationship to Eq. (6); we simply fix the same pattern to each $\mathsf{pa}(\ell)$. Namely, the parents of each label are the labels laying on the vertices above and to the left in the trellis structure (except, obviously, the vertices at the top and left of the trellis which form the border). A more linked structure will model dependence among more labels.

Hence, instead of trying to solve the NP hard structure discovery problem, we use a simple heuristic (label-frequency-based pairwise mutual information) to place the labels into a fixed structure (the trellis) in a sensible order: one that tries to maximize label dependence between parents and children. This ensures a good structure, which captures some of the main label dependencies, while maintaining scalability to a large number of labels and data. Namely, we employ an efficient *hill climbing* method to insert nodes into this trellis according to marginal dependence information, in a manner similar to the FS method in [1].

The process followed is outlined in Algorithm 1, to which we pass a pairwise matrix of mutual information, where

$$I(Y_\ell; Y_k) = \sum_{y_\ell \in \mathcal{Y}_\ell} \sum_{y_k \in \mathcal{Y}_k} p(y_\ell, y_k) \log \Big(\frac{p(y_\ell, y_k)}{p(y_\ell) p(y_k)} \Big).$$

Essentially, new nodes are progressively added to the graph based on the mutual information. Since the algorithm starts by placing a *random* label in the upper left corner vertex, a different trellis will be discovered for different random seeds. Each label y_ℓ is directly connected to a fixed number of parent labels in the directed graph (e.g., in Figure 5 each node has two parents in the first two graphs, and four in the third one) – except for the border cases where no parents are possible. The computational cost of this algorithm is $O(L^2)$, but due to the simple calculations involved, in practice it is easily able to scale up to tens of thousands of labels. Indeed, we show later in Section 5 that this method is fast and effective. Furthermore, it is possible to limit the complexity by searching for some number $< L$ of labels (e.g., building clusters of labels).

Given a proper user-defined parent pattern $\mathsf{pa}(\ell)$ (see Figure 5), we are ensured that the trellis obtained in Algorithm 1 is a directed acyclic graph. Hence, there is no need to check for cycles during construction, which is a very time consuming stage in many algorithms (e.g., in the PC algorithm, see Section 3.3). We can now employ probabilistic classifiers to construct an approximation to $p(\mathbf{y}|\mathbf{x})$ according to the directed graph. This approach is simply referred to as the *Classifier Trellis* (CT). Afterwards, we can either do inference greedily or via Monte Carlo sampling (see B for a detailed discussion of Monte Carlo methods).

Alternatively, note that we can interpret the trellis structure provided by Algorithm 1 in terms of an undirected graph, following the approach of Classifier Dependency Networks, described in Section 3.4. For example, in Figure 5 (middle), we would get (with directionality removed) $\mathsf{ne}(5) = \{2, 4, 6, 8\}$. This compares[6] to $\mathsf{pa}(5) = \{1, 2, 4\}$. We refer to this approach as the *Classifier Dependency Trellis* (CDT).

Both CT and CDT are outlined in Algorithm 2 and Algorithm 3 respectively. Some may argue that the undirected version is more powerful, since learning an undirected graph is typically easier than learning a directed graph that encodes causal relations. However, CDTconstructs an undirected graphical model where greedy inference cannot be implemented and we have to rely on (slower) Monte Carlo sampling methods in the test stage. This effect can be clearly noticed in Table 8.

[6]We did not add the diagonals in the set $\mathsf{ne}(5)$, since we wish that it be comparable in terms of the *number* of connections

Algorithm 1 Constructing a Classifier Trellis

input : $\boldsymbol{Y}$ (an $L \times N$ matrix of labels), W (width, default: $\sqrt{L}$), a parent-node function $\mathsf{pa}(\cdot)$
begin
  $\boldsymbol{I} \leftarrow$ LEARNDEPENDENCYMATRIX($\boldsymbol{Y}$) ;
  $S =$ SHUFFLE($\{1, \ldots, L\}$) ;
  $s_1 = S_1$;
  for $\ell = 2, \ldots, L$; **do**
    $S = S \setminus s_{\ell-1}$;
    $s_\ell = \text{argmax}_{k \in S} \sum_{j \in \mathsf{pa}(\ell)} I(y_j; y_k)$;
  end
end
output: the trellis structure, $\begin{bmatrix} s_1 & s_2 & \cdots & s_{W-1} & s_W \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ s_{L-W} & s_{L-W+1} & \cdots & s_{L-1} & s_L \end{bmatrix}$.

We henceforth assume that $y_\ell = y_{s_\ell}$.

Algorithm 2 Classifier Trellis (CT)

TRAIN($\mathcal{D}$) **begin**
  Find the directed trellis graph using Algorithm 1.
  Train classifiers $f_1, \ldots, f_L$, each taking *all parents* in the directed graph as additional features, such that

$$f_\ell(\mathbf{x}) \approx p(y_\ell | y_{\mathsf{pa}(\ell)}, \mathbf{x}).$$

end
TEST($\mathbf{x}^*$) **begin**
  for $\ell = \{s_1, \ldots, s_L\}$; **do**
    $\hat{y}_\ell = \text{argmax}_{y_\ell} \, p(y_\ell | \mathbf{y}_{\mathsf{pa}(\ell)}, \mathbf{x}^*)$
  end
  return $\hat{\mathbf{y}} = [\hat{y}_1, \ldots, \hat{y}_L]$
end

Finally, we will consider a simple ensemble method for CT, similar to that proposed in [19] or [25] to improve classifier chain methods: M CT classifiers, each built from a different random seed, where the final label decision is made by majority voting. This makes the training time and inference M times larger. We denote this method as ECT. A similar approach could be followed for the CDT (thus building an ECDT), but, given the higher computational cost of CDT during the test stage, we have concerns regarding the scalability of this approach, so we have excluded it from the simulations.

5. EXPERIMENTS

Firstly, in Section 5.1, we compare E/CT and CDT with some high-performance MLC methods (namely ECC, BCC, MCC) that were discussed in Section 3. We show that an imposed trellis structure can compete with fully-cascaded chains such as ECC and MCC, and discovered structures like those provided by BCC. Our approach based on trellis structures achieves similar MLC performance (or better in many cases) while presenting improved scalable properties and, consequently, significantly lower running times.

All the methods considered are listed in Table 3. In Table 4 we summarize their complexity. DN represents the input dimensions (the dataset-dependent number of features times number of instances); we use $M = 10$ ensemble methods and $T = 100$ Gibbs iterations for CDT. While this complexity is just an intuitive measure, the experimental results reported in Section 5.1 confirm that CT running times are indeed very close to IC.

Table 5 summarizes the collection of datasets that we use, of varied type and dimensions; most of them familiar to the MLC/MOC community [22, 3, 20].

The last two sets (Local400 and Local10k) are synthetically generated and they correspond to the localization problem

Algorithm 3 Classifier Dependency Trellis (`CDT`)

TRAIN($\mathcal{D}$) **begin**

Find the undirected trellis graph using Algorithm 1.

Train classifiers $f_1, \ldots, f_L$, each taking *all neighbouring labels* as additional features, such that

$$f_\ell(\mathbf{x}) \approx p(y_\ell | \mathbf{y}_{\mathsf{ne}}(\ell), \mathbf{x}).$$

end

TEST($\mathbf{x}^*$) **begin**

for $t = 1, \ldots, T_c, \ldots, T$; **do**

for $\ell = \text{SHUFFLE}\{1, \ldots, L\}$; **do**

$y_\ell^{(t)} \sim p(y_\ell | \mathbf{y}_{\mathsf{ne}}(\ell), \mathbf{x}^*)$

end

end

return $\hat{\mathbf{y}} = \frac{1}{T - T_c} \left[\sum_{T_c < t < T} y_1^{(t)}, \ldots, \sum_{T_c < t < T} y_L^{(t)} \right]$

end

where T_c, T: settling time (discarded samples) and total iterations.

Table 3: Methods tested in experiments.

Key	Description/Name	Reference
`IC`	Independent Classifiers	Sec. 1
`ECC`	Ensemble of 10 random `CC`s (majority vote per label)	Sec. 3, [20]
`MCC`	Best of 10 random `CC`	Sec. 3, [17]
`EBCC`	Ensemble of $\min(10, L)$ `BCC`s (discovered directed graph)	Sec. 3, [25]
`CT`	Classifier trellis, as in Figure 5, left (directed trellis)	Sec. 4, Alg. 2
`ECT`	Ensemble of 10 `CT`s	Sec. 4
`CDT`	Classifier dependency trellis, as in Figure 5, middle (but *undirected*); $T = 100, T_c = 10$	Sec. 4, Alg. 3

Table 4: Complexity per algorithm, roughly sorted by training complexity. DN represents the input dimensions (number of features times number of instances). Train complexity indicates roughly how many values are looked at by a classifier. Test complexity indicates how many (times) individual models are addressed as inference.

method	train complexity	test complexity
IC	$O(LDN)$	$O(L)$
CT	$O(LDN)$	$O(L)$
EBCC	$O(M \times LDN)$	$O(ML)$
CDT	$O(LDN)$	$O(TL)$
ECT	$O(M \times LDN)$	$O(ML)$
ECC	$O(M \times L[D+L]N)$	$O(ML)$

Table 5: A collection of datasets and associated statistics, where LC is *label cardinality*: the average number of labels relevant to each example.

	N	L	D	LC	Type
Music	593	6	72	1.87	audio
Scene	2407	6	294	1.07	image
Yeast	2417	14	103	4.24	biology
Medical	978	45	1449	1.25	medical/text
Enron	1702	53	1001	3.38	text
TMC2007	28596	22	500	2.16	text
MediaMill	43907	101	120	4.38	video
Delicious	16105	983	500	19.02	text
Local400	10000	400	30	12.84	localization
Local10k	50000	10000	30	25.78	localization

described in Section 5.2.

We use some standard metrics from the multi-label literature (see, e.g., [15]), namely,

$$\text{HAMMING SCORE} := \frac{1}{NL}\sum_{n=1}^{N}\sum_{j=1}^{L}\left[y_j^{(n)} = \hat{y}_j^{(n)}\right],$$

$$\text{EXACT MATCH} := \frac{1}{N}\sum_{n=1}^{N}\left[\mathbf{y}^{(n)} = \hat{\mathbf{y}}^{(n)}\right],$$

$$\text{ACCURACY}^{7} := \frac{1}{N}\sum_{i=1}^{N}\frac{|\mathbf{y}^{(n)} \wedge \hat{\mathbf{y}}^{(n)}|}{|\mathbf{y}^{(n)} \vee \hat{\mathbf{y}}^{(n)}|},$$

where $\left[A\right]$ is an identity function, returning 1 if condition A is true, whereas $\wedge$ and $\vee$ are the bitwise logical AND and OR operations, respectively.

5.1. Comparison of E/CT to other MLC methods

First of all, to confirm that our proposed hill climbing strategy can actually have a beneficial effect (i.e., an improvement over a random trellis), we do a 10×10 cross validation (CV) on the smaller datasets. Results are displayed in Table 6. A significant increase in performance can be seen, with a decrease in the standard deviation (especially relevant in the Scene dataset). This confirms that the proposed hill climbing strategy can help in optimizing performance and decreasing the sensitivity of CT wrt the initialization.

[7] Known commonly as *Jaccard Index* in information retrieval circles.

Table 6: Comparing CT accuracy (on Scene, Music) without (just a random trellis) and with our 'hill-climbing' (HC) method, over 10×10 CV.

	CT-random	CT-HC
Music	0.536 ± 0.042	0.541 ± 0.031
Scene	0.639 ± 0.210	0.678 ± 0.028

Then, we compare all the methods listed in Table 3 on all the datasets of Table 5. The results of predictive performance are displayed in Table 7, and running times can be seen in Table 8. On the small datasets ($L < 100$) we use Support Vector Machines as base classifiers, with fitted logistic models (as in [11]) for CDN (which requires probabilistic output for inference). As an alternative, other authors have used Logistic Regression directly (e.g., [3, 9]) due to its probabilistic output. In our experience, we obtain better and faster all-round performance with SVMs. Note that, for best accuracy, it is highly recommended to tune the base classifier. However, we wish to avoid this "dimension" and instead focus on the multi-label methods. On the larger datasets (where $L > 100$) we instead use Stochastic Gradient Descent (SGD), with a maximum of only 100 epochs, to deal with the scale presented by these large problems. All our methods are implemented and made available within the Meka framework;[8] an open-source multi-output learning framework based on the Weka machine learning framework [10]. The SMO SVM and SGD implementations pertain to Weka.

Results confirm that both ECT and CT are very competitive in terms of performance and running time. Using the Hamming score as figure of merit and given the running times reported, CT is clearly superior to the rest of methods. Note that Table 8 shows that CT's running times are close to IC, namely the method that neglects all statistical dependency between labels. With respect to exact match and accuracy performance results, which are measures oriented to the recovery of the whole set of labels, ECT and CT achieve very competitive results with respect to ECC and MCC, which are high-complexity methods that model the full-chain of labels. Table 8 reports training and test average times computed for the different MLC methods. We also include explicitly the number L of labels per dataset. Note also that even though ECT considers a set of 10 possible random initializations, it does not significantly improve the performance of CT (a single initialization) for most cases, which suggest that the hill climbing strategy makes the CT algorithm quite robust with respect to initialization. Regarding scalability, note that the computed train/running times for CT scale roughly linearly with the number of labels L. For instance, in the MediaMill dataset $L = 101$ while in Delicious L is approximately one order of magnitude higher, $L = 983$. As we can observe, CT running times are multiplied approximately by a factor of 10 between both datasets. The same conclusions can be drawn also for the largest datasets, compare for instance running times between Local400 and Local10k. Finally, CDT, our scalable modification of CDN, shows worse performance than CT while it requires larger test running times.

In order to illustrate the most significant statistical differences between all methods, in Figure 6 we include the results of the Nemenyi test based on Table 7 and Table 8. However, note that here we excluded the two rows with DNFs. The Nemenyi test [6] rejects the null hypothesis if the average rank difference is greater than the *critical distance* $q_p\sqrt{N_A(N_A+1)/6N_D}$ over N_A algorithms and N_D datasets, and q_p according to the q table for some p value (we use $p = 0.90$). Any method with a rank greater than another method by at least the critical distance, is considered statistically better. In Figure 6, for each method, we place a bar spanning from the average rank of the method, to this point *plus* the critical distance. Thus, any pair of bars that *do not* overlap correspond to methods that are statistically different in terms of performance. Note that, regarding both training and test running times, CT overlaps considerably with IC, whereas other methods such as ECC and MCC need significantly more training time. We can say that ECC and ECT are statistically stronger than IC and CDT, but not so wrt exact match. CT performs particularly well on the Hamming score, indicating that error propagation is limited, compared to other CC methods.

In the following section we present the framework behind the localization datasets Local400 and Local10k in the tables presented above.

[8] http://meka.sourceforge.net

Table 7: Predictive performance and dataset-wise (rank). DNF = Did Not Finish (within 24 hours or 2 GB memory).

Accuracy

Dataset	IC	ECC	MCC	EBCC	CT	ECT	CDT
Music	0.483 (7)	0.572 (2)	0.568 (4)	0.566 (5)	0.577 (1)	0.571 (3)	0.505 (6)
Scene	0.571 (7)	0.684 (2)	0.685 (1)	0.618 (4)	0.602 (6)	0.666 (3)	0.604 (5)
Yeast	0.502 (6)	0.538 (2)	0.534 (4)	0.535 (3)	0.533 (5)	0.541 (1)	0.438 (7)
Medical	0.699 (7)	0.733 (3)	0.721 (5)	0.731 (4)	0.755 (2)	0.769 (1)	0.704 (6)
Enron	0.406 (5)	0.448 (1)	0.403 (6)	0.441 (3)	0.409 (4)	0.443 (2)	0.310 (7)
TMC07	0.614 (5)	0.645 (1)	0.619 (4)	0.628 (3)	0.613 (6)	0.633 (2)	0.601 (7)
MediaMill	0.379 (2)	0.350 (5)	0.349 (6)	0.375 (3)	0.391 (1)	0.344 (7)	0.374 (4)
Delicious	0.122 (3)	DNF	0.121 (5)	DNF	0.127 (2)	0.157 (1)	0.122 (3)
Local400	0.536 (7)	0.625 (1)	0.583 (3)	0.578 (4)	0.542 (6)	0.587 (2)	0.559 (5)
Local10k	0.125 (4)	DNF	DNF	0.175 (1)	0.133 (3)	0.166 (2)	0.122 (5)
avg rank	5.30	2.12	4.22	3.33	3.60	2.40	5.50

Hamming Score

Dataset	IC	ECC	MCC	EBCC	CT	ECT	CDT
Music	0.785 (6)	0.795 (3)	0.789 (5)	0.800 (1)	0.798 (2)	0.795 (3)	0.768 (7)
Scene	0.886 (4)	0.892 (1)	0.892 (1)	0.886 (4)	0.884 (6)	0.891 (3)	0.871 (7)
Yeast	0.800 (1)	0.789 (4)	0.794 (2)	0.787 (5)	0.791 (3)	0.786 (6)	0.719 (7)
Medical	0.988 (4)	0.988 (4)	0.989 (2)	0.988 (4)	0.990 (1)	0.989 (2)	0.986 (7)
Enron	0.943 (1)	0.940 (4)	0.942 (3)	0.939 (5)	0.943 (1)	0.939 (5)	0.922 (7)
TMC07	0.947 (2)	0.948 (1)	0.947 (2)	0.946 (5)	0.947 (2)	0.946 (5)	0.937 (7)
MediaMill	0.965 (2)	0.947 (7)	0.958 (4)	0.954 (5)	0.966 (1)	0.951 (6)	0.965 (2)
Delicious	0.982 (1)	DNF	0.981 (4)	DNF	0.982 (1)	0.981 (4)	0.982 (1)
Local400	0.968 (3)	0.969 (1)	0.967 (6)	0.968 (3)	0.969 (1)	0.968 (3)	0.962 (7)
Local10k	0.968 (1)	DNF	DNF	0.968 (1)	0.968 (1)	0.968 (1)	0.968 (1)
avg rank	2.50	3.12	3.22	3.67	1.90	3.80	5.30

Exact Match

Dataset	IC	ECC	MCC	EBCC	CT	ECT	CDT
Music	0.252 (7)	0.327 (1)	0.292 (5)	0.302 (4)	0.312 (2)	0.312 (2)	0.257 (6)
Scene	0.491 (7)	0.579 (2)	0.638 (1)	0.516 (5)	0.542 (4)	0.557 (3)	0.503 (6)
Yeast	0.160 (5)	0.190 (3)	0.212 (1)	0.150 (6)	0.198 (2)	0.169 (4)	0.067 (7)
Medical	0.614 (4)	0.612 (5)	0.634 (3)	0.612 (5)	0.670 (1)	0.655 (2)	0.598 (7)
Enron	0.121 (3)	0.112 (5)	0.126 (1)	0.114 (4)	0.123 (2)	0.112 (5)	0.067 (7)
TMC07	0.330 (4)	0.342 (2)	0.345 (1)	0.316 (6)	0.341 (3)	0.317 (5)	0.263 (7)
MediaMill	0.055 (2)	0.034 (5)	0.053 (3)	0.019 (6)	0.058 (1)	0.007 (7)	0.052 (4)
Delicious	0.003 (3)	DNF	0.006 (1)	DNF	0.004 (2)	0.002 (5)	0.003 (3)
Local400	0.064 (4)	0.108 (2)	0.129 (1)	0.059 (6)	0.079 (3)	0.063 (5)	0.029 (7)
Local10k	0.000 (1)	DNF	DNF	0.000 (1)	0.000 (1)	0.000 (1)	0.000 (1)
avg rank	4.00	3.12	1.89	4.78	2.10	3.90	5.50

Table 8: Time results (seconds). DNF = Did Not Finish (within 24 hours or 2 GB memory).

Training Time

Dataset	L	IC	ECC	MCC	EBCC	CT	ECT	CDT
Music	6	1 (3)	4 (6)	4 (7)	2 (5)	0 (1)	2 (4)	1 (2)
Scene	6	3 (1)	10 (5)	28 (7)	7 (4)	3 (3)	10 (6)	3 (2)
Yeast	14	11 (3)	53 (6)	79 (7)	45 (5)	5 (2)	26 (4)	5 (1)
Medical	45	4 (2)	19 (6)	67 (7)	17 (4)	3 (1)	19 (5)	5 (3)
Enron	53	51 (3)	207 (6)	734 (7)	95 (4)	24 (1)	100 (5)	37 (2)
TMC07	22	11402 (2)	48019 (6)	73433 (7)	34559 (4)	10847 (1)	44986 (5)	13547 (3)
MediaMill	101	42 (1)	347 (6)	1121 (7)	238 (5)	45 (2)	219 (4)	55 (3)
Delicious	983	468 (1)	DNF	18632 (5)	DNF	529 (2)	2791 (4)	599 (3)
Local400	400	2 (1)	15 (5)	57 (7)	8 (3)	2 (2)	9 (4)	31 (6)
Local10k	10^4	3 (1)	DNF	DNF	13 (4)	3 (2)	15 (5)	3 (3)
avg rank		1.80	5.75	6.78	4.22	1.70	4.60	2.80

Test Time

Dataset	L	IC	ECC	MCC	EBCC	CT	ECT	CDT
Music	6	0 (3)	1 (7)	0 (1)	0 (5)	0 (4)	0 (2)	0 (6)
Scene	6	0 (1)	1 (5)	0 (2)	0 (4)	0 (3)	2 (6)	7 (7)
Yeast	14	1 (2)	3 (5)	3 (6)	1 (3)	0 (1)	2 (4)	8 (7)
Medical	45	4 (2)	28 (6)	9 (3)	26 (5)	2 (1)	22 (4)	141 (7)
Enron	53	4 (1)	112 (6)	8 (3)	19 (4)	4 (2)	45 (5)	310 (7)
TMC07	22	7 (2)	50 (5)	3 (1)	42 (4)	7 (3)	76 (6)	534 (7)
MediaMill	101	15 (1)	211 (5)	31 (2)	143 (4)	32 (3)	972 (6)	5469 (7)
Delicious	983	167 (1)	DNF	322 (3)	DNF	207 (2)	7532 (4)	31985 (5)
Local400	400	1 (1)	17 (6)	3 (3)	9 (4)	1 (2)	17 (5)	398 (7)
Local10k	10^4	4 (2)	DNF	DNF	18 (3)	4 (1)	39 (4)	4228 (5)
avg rank		1.60	5.62	2.67	4.00	2.20	4.60	6.50

5.2. A Structured Output Prediction Problem

We investigate the application of CT (and the other MLC methods) to a type of structured output prediction problem: segmentation for localization. In this section we consider a localization application using light sensors, based on the real-world scenario described in [16], where a number of light sensors are arranged around a room for the purpose of detecting the location of a person. We take a 'segmentation' view of this problem, and use synthetic models (which are based on real sensor data) to generate our own observations, thus creating a semi-synthetic dataset, which allows us to easily control the scale and complexity. Figure 7 shows the scenario. It is a top-down view of a room with light sensors arranged around the edges, one light source (a window, drawn as a thin rectangle) on the bottom edge and four targets. Note that targets can only be detected if they come between a light sensor and the light source, so the target in the lower right corner is undetectable.

We divide the scenario into $L = W \times W$ square 'tiles', representing $Y_1, \ldots, Y_L$. Given an instance n,

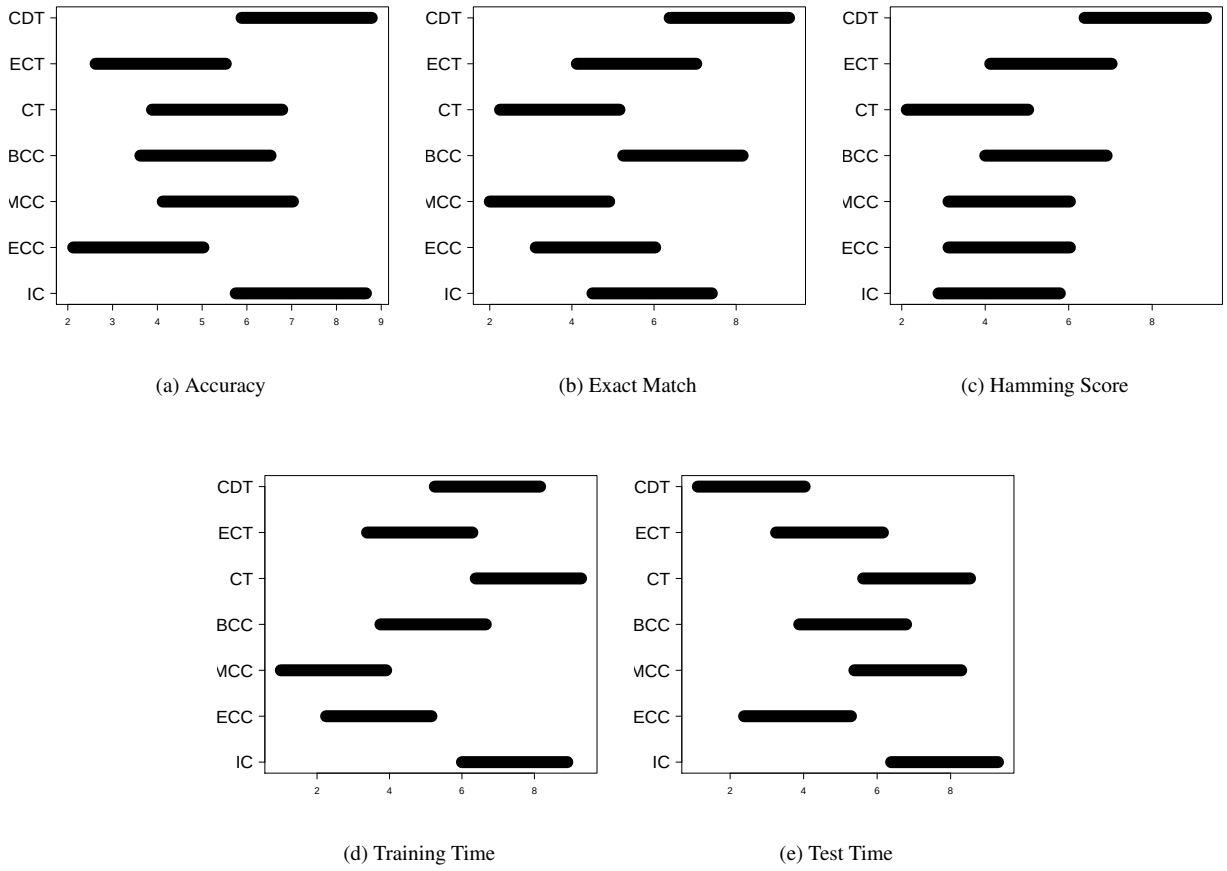
$$\mathbf{y}^{(n)} = \begin{bmatrix} y_{1,1}^{(n)} & y_{1,2}^{(n)} & \cdots & \cdots & y_{1,W}^{(n)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ y_{W,1}^{(n)} & y_{W,2}^{(n)} & \cdots & \cdots & y_{W,W}^{(n)} \end{bmatrix},$$

where $y_{i,j}^{(n)} = 1$ if the i,j-th tile (i.e., pixel) is active, and $y_{i,j}^{(n)} = 0$ otherwise, with $i, j \in \{1, \ldots, W\}$. For the n-th instance we have binary sensor observations $\mathbf{x}^{(n)} = [x_1^{(n)}, \ldots, x_D^{(n)}]$, where $x_d = 1$ if the d-th sensor detects an object inside its 'detection zone' (shown in colors in Figure 7). Otherwise, $x_d = 0$.

5.2.1. Sensor Model

Consider, for simplicity, a specific instance $\mathbf{y} = \{y_{i,j}\}_{i,j=1}^{W}$ (in order to avoid here the use of the super index n). Moreover, let us denote as $\mathbf{s}_d = [s_{1,d}, s_{2,d}]$ the position of the d-th sensor, and Z_d the triangle of vertices $\mathbf{s}_d$, $\mathbf{l}_1 = [2.5, 0]$ and $\mathbf{l}_2 = [7.5, 0]$ (the corners of the light source). This triangle Z_d is the "detection zone" of the d-th sensor. Now, we define the indicator variable

$$z_{d,i,j} = 1 \quad \text{if } \left(i - \frac{1}{2}, j - \frac{1}{2}\right) \in Z_d, \quad z_{d,i,j} = 0 \quad \text{if } \left(i - \frac{1}{2}, j - \frac{1}{2}\right) \notin Z_d,$$



(a) Accuracy (b) Exact Match (c) Hamming Score

(d) Training Time (e) Test Time

Fig. 6: Results of Nemenyi test, based on Table 7 and Table 8, If methods' bars overlap, they can be considered statistically indifferent. The graphs based on time should be interpreted such that higher rank (more to the left) corresponds to slower (i.e., less desirable) times.

where $\left(i-\frac{1}{2}, j-\frac{1}{2}\right)$ is the middle point of the (i,j)-th pixel (tile), whose vertices are (i,j), $(i-1,j-1)$, $(i-1,j)$ and $(i,j-1)$ (for $i,j=2,\ldots,W$). Next, we define the variable

$$c_d = \sum_{i=1}^{W}\sum_{j=1}^{W} y_{i,j} z_{d,i,j}, \tag{10}$$

which corresponds to the number of active tiles/pixels inside the triangle Z_d associated to the d-th sensor. The likelihood function for the d-th sensor is then given by

$$p(x_d=1|\mathbf{y}) = p(x_d=1|c_d) = \begin{cases} \epsilon_2, & c_d=0; \\ 1-\epsilon_1, & c_d=1; \\ 1-\epsilon_1\exp[-0.1(c_d-1)], & c_d>1; \end{cases} \tag{11}$$

and $p(x_d=0|\mathbf{y}) = 1-p(x_d=1|\mathbf{y})$; where $\epsilon_1 = p(x_d=0|c_d=1) = 0.15 < 0.5$ is the false *negative* rate and $\epsilon_2 = p(x_d=1|c_d=0) = 0.01 < 0.5$ is the false *positive* rate.

5.2.2. Generation of Artificial Data

Figure 7 shows a low dimensional scenario ($L=100, W=10$), for the purpose of a clear illustration, but we consider datasets with much higher levels of segmentation (namely Local400, where $L=400$, and Local10k, where $L=10,000$ – see Table 5)

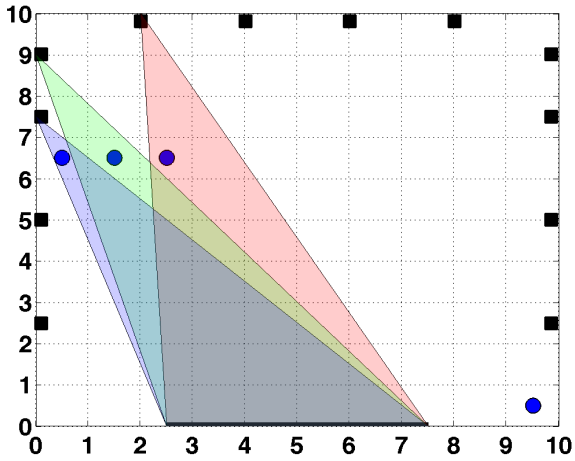


Fig. 7: An $L = 10 \times 10 = 100$ tile localization scenario. $D = 12$ light sensors are arranged around the edges of the scenario at coordinates $\mathbf{s}_1, \ldots, \mathbf{s}_D$, and there is a light source between points $\mathbf{l}_1 = [2.5, 0]$ and $\mathbf{l}_2 = [7.5, 0]$ (shown as a thick black line) on the horizontal axis. In this example, three observations are positive ($x_d = 1$). Note that the object in the bottom-right tile ($y_L = 1$ in this case) cannot be detected.

to compare the performance of several MOC techniques on this problem. Given a scenario with $L = W \times W$ tiles, D sensors and N observations, we generate the synthetic data, $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})_{n=1}^{N}$, as follows:

1. Start with an 'empty' $\mathbf{y}$, i.e., $y_{i,j} = 0$ for $i, j = 1, \ldots, W$.
2. Set $y_{i,j} = 1$ for relevant tiles to create a rectangle of width $W/8$ and height 2 starting from some random point $y_{i,j}$.
3. Create a 2×2 square in the corner *furthest from* the rectangle.
4. Generate the observations according to Eq. (11).
5. Add dynamic noise in $\mathbf{y}$ by flipping $L/100$ pixels uniformly at random.

Any MLC method can be applied to this problem, to infer the binary vector $\mathbf{y}^*$, which encodes the presence of blocking-light elements in the room, given the vector $\mathbf{x}^*$ of measurements from the light sensors. Finally, we also consider that each sensor provides M observations $\{x_{d,k}\}_{k=1}^{M} \subset \{0, 1\}^M$ given the same $\mathbf{y}$.

5.2.3. Maximum A Posteriori (MAP) Estimator

Given the likelihood function of Eq. (11), and considering a uniform prior over each variable c_d, the posterior w.r.t. the d-th triangle is

$$p(c_d|x_d) \propto p(x_d|c_d) = p(x_d|\mathbf{y}).$$

If we also assume independency in the received measurements, the posterior density $p(c_1, \ldots, c_D|\mathbf{x})$ can be expressed as follows

$$p(c_1, \ldots, c_D|\mathbf{x}) = \prod_{d=1}^{D} p(c_d|\mathbf{x}) \propto \prod_{d=1}^{D} p(x_d|c_d) = \prod_{d=1}^{D} p(x_d|\mathbf{y}). \tag{12}$$

We are interested in studying $p(y_{i,j}|\mathbf{x})$ for $1 \leq i, j \leq W$, but we can only compute the posterior distribution of the variables $\{c_1, \ldots c_D\}$, which depend on $y_{i,j}$ through Eq. (10). Making inference directly on $y_{i,j}$ using the posterior distribution $p(c_1, \ldots, c_d|\mathbf{x})$ is not straightforward. Let us address the problem in two steps. First, the measurements received by each sensor, x_d, can be considered as Bernoulli trials: if $c_d = 0$, then $x_d = 1$ with probability $\theta_d = \epsilon_2$; if $c_d \geq 1$, then $x_d = 1$ with

Algorithm 4 MAP inference using the sensor model

input : $\{x_{d,k}\}_{k=1,d=1}^{M,D}$ (measurements), $\{\mathbf{s}_d\}_{d=1}^{D}$ (sensor positions), $\mathbf{l}_1$ and $\mathbf{l}_2$ (light source location).

begin

1. Initialize $\hat{y}_{i,j} = 0.5$ for $i, j = 1, \ldots, W$.
2. **for** $d = 1, \ldots, D$; **do**
   (a) Calculate the detection triangle Z_d.
   (b) If $\hat{\theta}_d = \frac{1}{M}\sum_{k=1}^{M} x_{d,k} \leq 0.5$, then set $\hat{c}_d = 0$.
   (c) Otherwise, if $\hat{\theta}_d = \frac{1}{M}\sum_{k=1}^{M} x_{d,k} > 0.5$, then set $\hat{c}_d = 1$.
   (d) If $\hat{c}_d = 0$, then set $\hat{y}_{i,j} = 0$ for all $i, j \in Z_d$.

   end
3. For all d such that $\hat{c}_d = 1$ and for all $i, j \in Z_d$, check if the decision is still $\hat{y}_{i,j} = 0.5$. Then, set $\hat{y}_{i,j} = 1$.
4. The remaining tiles with $\hat{y}_{i,j} = 0.5$ correspond to "shadow" zones, where we leave $\hat{y}_{i,j} = 0.5$.

end

output: $\hat{y}_{i,j}$ for $i, j = 1, \ldots, W$.

Table 9: Results using Algorithm 4 with $D = 30$ sensors.

Measure	$W = 20$	$W = 100$
Accuracy	0.523	0.141
Hamming score	0.857	0.795
Times (total, s)	1	3

success probability $\theta_d = 1 - \epsilon_1$. Now, given M measurements for each sensor $\{x_{d,k}\}_{k=1}^{M} \subset \{0,1\}^M$ and uniform prior density over θ_d, the MAP estimator of θ_d is given by

$$\hat{\theta}_d = \frac{1}{M}\sum_{k=1}^{M} x_{d,k}.$$

Then, if $\hat{\theta}_d \leq 0.5$ we decide $\hat{c}_d = 0$. Otherwise, if $\hat{\theta}_d > 0.5$, we estimate $\hat{c}_d \geq 1$. Considering a uniform prior over the pixels $y_{i,j}$, a simple procedure to estimate $\mathbf{y}$ from $\{\hat{c}_1, \ldots \hat{c}_D\}$ is the one described in Algorithm 4.

5.2.4. Classifier Trellis vs MAP Estimator

Results for `CT` are already given in Table 7 (predictive performance) and Table 8 (running time). Results in Table 7 illustrate the robustness of the `CT` algorithm to address multi-output classification in several scenarios. Beyond the training set, no further knowledge about the underlying model is needed to achieve remarkable classification performance. To emphasize this property of `CT`, we now compare it to the MAP estimator presented above, which exploits a perfect knowledge of the sensor model.

Table 9 shows the results using Algorithm 4 with $D = 30$ sensors and different values of W (i.e., the grid precision). The corresponding results obtained by CT are provided in Table 10. A detailed discussion of these results is provided at the end of the next Section. However, let us remark that increasing the number of tiles (i.e., W) for a given number of sensors D makes the problem harder, as a finer resolution is sought. This explains the decrease in performance seen in the tables as W increases.

6. DISCUSSION

As in most of the multi-label literature, we found that independent classifiers consistently under-perform, thus justifying the development of more complex methods to model label dependence. However, in contrary to what much of the multi-label

Table 10: Results using CT ($D = 30$ sensors).

Measure	$W = 20$	$W = 100$
Accuracy	0.542	0.133
Hamming score	0.969	0.968
Time (total, s)	3	7

literature suggests, greater investments in modelling label dependence do not always correspond to greater returns. In fact, it appears that many methods from the literature have been over-engineered. Our small experiment in Table 2 suggests that none of the approaches we investigated were particularly dominant in their ability to uncover structure with respect to predictive performance. Indeed, our results indicate that none of the techniques is significantly better than another. Using ECC is a 'safe bet' in terms of high accuracy, since it models long term dependencies with a fully cascaded chain; also noted previously (e.g,. [20, 3]). In terms of EBCC (for which we elected to represent methods that uncover a structure), there was no clear advantage over the other methods, and surprisingly also no clear difference between searching for a structure based on marginal dependence versus conditional label dependence. This makes it more difficult to justify computationally complex expenditures for modelling dependence on the basis of improved accuracy; particularly so for large datasets, where the scalability is crucial.

We presented a classifier trellis (CT) as an alternative to methods that model a full chain (as MCC or ECC) or methods that unravel the label graphical model structure from scratch, such as BCC. Our approach is systematic, we consider a fixed structure in which we place the labels in an ordered procedure according to easily computable mutual information measures, see Algorithm 1. An ensemble version of CT performs particularly well on exact match but, surprisingly, it does not perform much stronger than CT as we expected in the beginning. It does not perform as strong overall as ECC (although there is no statistically significant difference), but is much more scalable, as indicated in Table 4.

The CT algorithm then emerges as a powerful MLC algorithm, able to excellent performance (specially in terms of average number of successfully classified labels) with near IC running times. Through the Nemenyi test, we have shown the statistical similitude between the classification outputs of (E)CT and MCC/ECC, proving that our approach based on the classifier trellis captures the necessary inter-label dependencies to achieve high performance classification. Moreover, we have not analyzed yet the impact that the trellis structure chosen has in the CT performance. In future work, we intend to experiment with trellis structures with different degrees of connectedness.

7. ACKNOWLEDGEMENTS

This work was supported by the Aalto University AEF research programme; by the Spanish government's (projects projects 'COMONSENS', id. CSD2008-00010, 'ALCIT', id. TEC2012-38800-C03-01, 'DISSECT', id. TEC2012-38058-C03-01); by Comunidad de Madrid in Spain (project 'CASI-CAM-CM', id. S2013/ICE-2845); and by and by the ERC grant 239784 and AoF grant 251170.

8. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets in items in large databases. In *Proc. of ACM SIGMOD 12*, pages 207–216, 1993.

[2] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[3] Weiwei Cheng, Krzysztof Dembczyński, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML '10: 27th International Conference on Machine Learning*, Haifa, Israel, June 2010. Omnipress.

[4] Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors. *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*. ACM, 2009.

[5] Krzysztof Dembczyński, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. On label dependence and loss minimization in multi-label classification. *Mach. Learn.*, 88(1-2):5–45, July 2012.

[6] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[7] Nadia Ghamrawi and Andrew McCallum. Collective multi-label classification. In *CIKM '05: 14th ACM international Conference on Information and Knowledge Management*, pages 195–200, New York, NY, USA, 2005. ACM Press.

[8] Anna Goldenberg and Andrew Moore. Tractable learning of large bayes net structures from sparse data. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 44–, New York, NY, USA, 2004. ACM.

[9] Yuhong Guo and Suicheng Gu. Multi-label classification using conditional dependency networks. In *IJCAI '11: 24th International Conference on Artificial Intelligence*, pages 1300–1305. IJCAI/AAAI, 2011.

[10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Reutemann Peter, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[11] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998.

[12] Markus Kalisch and Peter Bühlmann. Estimating high-dimensional directed acyclic graphs with the pc-algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

[13] Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Learning and inference in probabilistic classifier chains with beam search. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7523, pages 665–680. Springer, 2012.

[14] L. Ladick, C. Russell, P. Kohli, and P. H S Torr. Associative hierarchical crfs for object class image segmentation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 739–746, Sept 2009.

[15] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Deroski. An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, 45(9):3084–3104, September 2012.

[16] Jesse Read, Katrin Achutegui, and Joaquin Miguez. A distributed particle filter for nonlinear tracking in wireless sensor networks. *Signal Processing*, 98:121–134, 2014.

[17] Jesse Read, Luca Martino, and David Luengo. Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, 47(3), 2014.

[18] Jesse Read, Bernhard Pfahringer, and Geoff Holmes. Multi-label classification using ensembles of pruned sets. In *ICDM'08: Eighth IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2008.

[19] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. In *ECML '09: 20th European Conference on Machine Learning*, pages 254–269. Springer, 2009.

[20] Jesse Read, Bernhard Pfahringer, Geoffrey Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

[21] Hongyu Su and Juho Rousu. Multilabel classification through random graph ensembles. In *Asian Conference on Machine Learning (ACML)*, pages 404–418, 2013.

[22] Grigorios Tsoumakas and Ioannis P. Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *ECML '07: 18th European Conference on Machine Learning*, pages 406–417. Springer, 2007.

[23] Celine Vens and Fabrizio Costa. Random forest based feature induction. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, pages 744–753, Washington, DC, USA, 2011. IEEE Computer Society.

[24] Raanan Yehezkel and Boaz Lerner. Bayesian network structure learning by recursive autonomy identification. *Journal of Machine Learning Research*, 10:1527–1570, 2009.

[25] Julio H. Zaragoza, Luis Enrique Sucar, Eduardo F. Morales, Concha Bielza, and Pedro Larrañaga. Bayesian chain classifiers for multidimensional classification. In *24th International Conference on Artificial Intelligence (IJCAI '11)*, 2011.

[26] Min-Ling Zhang and Kun Zhang. Multi-label learning by exploiting label dependency. In *KDD '10: 16th ACM SIGKDD International conference on Knowledge Discovery and Data mining*, pages 999–1008. ACM, 2010.

[27] Min-Ling Zhang and Zhi-Hua Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints):1, 2013.

A. GRAPHICAL MODEL STRUCTURE LEARNING: FS VS. LEAD

In order to delve deeper into the issue of structure learning, we generated a synthetic dataset, where the underlying structure is known. The synthetic generative model is as follows. For the feature vector, we consider a D-dimensional independent Gaussian vector $\mathbf{x} \in \mathbb{R}^D$, where $x_d \sim \mathcal{N}(0,1)$ for $d = 1, \ldots, D$. Let $\mathbf{w}_\ell$ ($\ell = 1, \ldots, L$) be a binary D-dimensional vector containing exactly T ones (and thus $D - T$ zeros), and let us assume that we have a directed acyclic graph between the labels in which each label has at most one parent. Both the vectors $\mathbf{w}_\ell$ and the dependency label graph are generated uniformly at random. Given the value of its parent label, $y_{\mathsf{pa}(\ell)} \in \{-1, 1\}$, the following probabilistic model is used to generate the ℓ-th label y_ℓ:

$$y_\ell = \begin{cases} +1, & T^{-1/2}\mathbf{w}_\ell^\top \mathbf{x} + \epsilon_\ell \geq \delta; \\ -1, & \text{otherwise}; \end{cases} \tag{13}$$

where δ is a real constant and $\epsilon_\ell \sim \mathcal{N}(\alpha\, y_{\mathsf{pa}(\ell)}, \sigma^2)$, with $\alpha \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$. Note that, according to the model, $y_\ell | y_{\mathsf{pa}(\ell)}$ is a Bernoulli random variable that takes value 1 with average probability

$$P(y_\ell = +1 | y_{\mathsf{pa}(\ell)} = +1) = Q\left(\frac{\delta - \alpha\, y_{\mathsf{pa}(\ell)}}{\sqrt{1+\sigma^2}}\right),$$

where $Q(x) = 1 - \Phi(x)$ and $\Phi(x)$ is the cumulative distribution function of the normal Gaussian distribution. Consequently, with α and σ^2 we control the likelihood of y_ℓ being equal to its parent $y_{\mathsf{pa}(\ell)}$, thus modulating the complexity of inferring such dependencies by using the FS and LEAD methods.

In Figure 8 we show three examples of synthetically generated datasets, in terms of their ground truth structure and the structure discovered using the FS and LEAD methods, for three different scenarios: 'easy' ($\alpha = 1, \sigma^2 = 1$), 'medium' ($\alpha = 0.5$, $\sigma^2 = 2$), and 'hard' ($\alpha = 0.25$, $\sigma^2 = 5$) datasets. Recall that we use a *mutual information* matrix for both methods, with the difference being that the LEAD matrix is based on the *error frequencies* rather than the label frequencies. Visually it appears that both FS and LEAD are able to discover the original structure, relative to the difficulty of the dataset. There appears to be a small improvement of FS over LEAD. This is confirmed in a batch analysis using the F-measure of 10 random datasets of random difficulty ranging between 'easy' and 'hard': FS gets 0.278 and LEAD gets 0.263. A more in depth comparison, taking into account varying numbers of labels and features, is left for future work.

B. APPROXIMATE INFERENCE VIA MONTE CARLO

A better understanding of the MLC/MOC approaches described in Section 3 and the novel scheme introduced in this work can be achieved by describing the Monte Carlo (MC) procedures used to perform approximate inference over the graphical models constructed to approximate $p(\mathbf{y}|\mathbf{x})$.

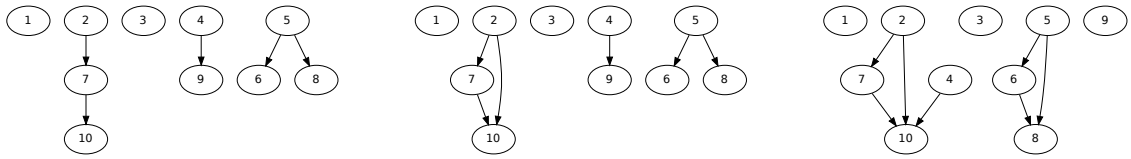
Given a probabilistic model for the conditional distribution $p(\mathbf{y}|\mathbf{x})$ and a new test input $\mathbf{x}^*$, the goal of an MC scheme is generating samples from $p(\mathbf{y}|\mathbf{x}^*)$ that can be used to estimate its mode (which is the MAP estimator of $\mathbf{y}$ given $\mathbf{x}^*$), the marginal distribution per label (i.e., $p(y_\ell|\mathbf{x}^*)$ for $\ell = 1 \ldots, L$) or any other relevant statistical function of the data.

B.1. Bayesian networks

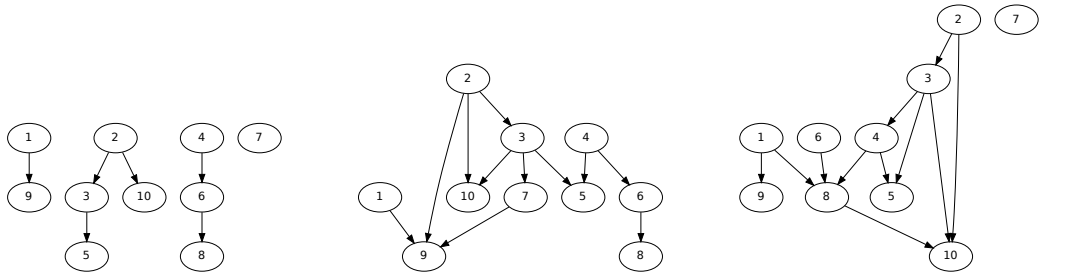
In a directed acyclic graphical model, the probabilistic dependencies between variables are ordered. For instance, in the CC scheme $p(\mathbf{y}|\mathbf{x})$ factorizes according to Eq. (3). If it is possible to draw samples directly from each conditional density $p(y_\ell|y_{1:\ell-1}, \mathbf{x})$, then exact sampling can be performed in a simple manner. For $i = 1, \ldots, N_s$ (where N_s is the desired number of samples), repeat

$$\begin{aligned} y_1^{(i)} &\sim p(y_1|\mathbf{x})), \\ y_2^{(i)} &\sim p(y_2|y_1^{(i)}, \mathbf{x})), \\ &\vdots \\ y_L^{(i)} &\sim p(y_L|y_{1:L-1}^{(i)}, \mathbf{x}). \end{aligned}$$

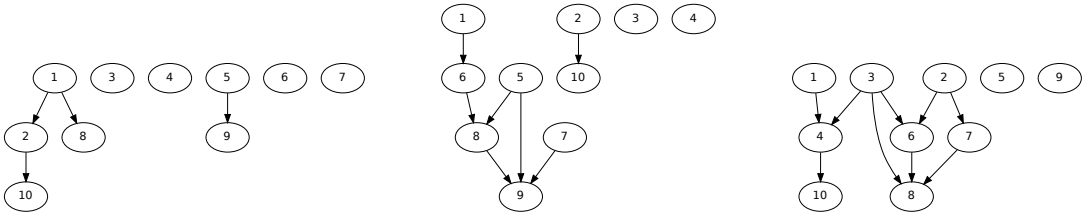
For Bayesian networks that are not fully-connected, as in BCC, the procedure is similar. Each sampled vector, $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \ldots, y_L^{(i)}]$ for $i = 1, \ldots, N_s$, is obtained by drawing each individual component independently as $y_\ell^{(i)} \sim p(y_\ell|\mathbf{y}_{\mathsf{pa}(\ell)}^{(i)})$ for $\ell = 1, \ldots, L$.



(a) Ground Truth (b) FS (c) LEAD



(d) Ground Truth (e) FS (f) LEAD



(g) Ground Truth (h) FS (i) LEAD

parameter	easy	medium	hard
α	1	0.5	0.25
σ^2	1	2	5

Fig. 8: Ground-truth graphs of the synthetic dataset (left) – easy, medium, and hard according to the table – and their reconstruction found by the FS strategy [8] (middle) and LEAD strategy [26] (right).

B.2. Markov networks

In an undirected graphical model (like that of a CDN), exact sampling is generally unfeasible. However, a Markov Chain Monte Carlo (MCMC) technique that is able to generate samples from the target density $p(\mathbf{y}|\mathbf{x})$ can be implemented. Within this class, *Gibbs Sampling* is often the most adequate approach. Let us assume that the conditional distribution $p(\mathbf{y}|\mathbf{x})$ factorizes according to an undirected graphical model as in Eq. (7). Then, from an initial configuration $\mathbf{y}^{(0)} = [y_1^{(0)}, \ldots, y_L^{(0)}]$, repeat for $t = 1, \ldots, T$:

$$\begin{aligned} y_1^{(t)} &\sim p(y_1|y_1^{(t-1)}, y_2^{(t-1)}, ..., y_L^{(t-1)}, \mathbf{x}), \\ y_2^{(t)} &\sim p(y_2|y_1^{(t)}, y_2^{(t-1)}, ..., y_L^{(t-1)}, \mathbf{x}), \\ &\vdots \\ y_L^{(t)} &\sim p(y_L|y_1^{(t)}, y_2^{(t)}, ..., y_L^{(t-1)}, \mathbf{x}), \end{aligned}$$

where each label can be sampled by conditioning just on the neighbors in the graph, as seen from Eq. (9). Thus,

$$y_\ell^{(t)} \sim p(y_\ell|\{y_u^{(t)} : y_u \in \mathbf{y}_{\mathsf{ne}(\ell)}, u < \ell\}, \{y_m^{(t-1)} : y_m \in \mathbf{y}_{\mathsf{ne}(\ell)}, m > \ell\}, \mathbf{x}),$$

which can be simply denoted as $y_\ell^{(t)} \sim p(y_\ell|\mathbf{y}_{\mathsf{ne}(\ell)}^{(t)})$, with $\mathbf{y}_{\mathsf{ne}(\ell)}^{(t)}$ denoting the state of the neighbors of y_ℓ at time t.

Following this approach, the state of the chain $\mathbf{y}^{(t)} = [y_1^{(t)}, \ldots, y_L^{(t)}]$ can be considered a sample from $p(\mathbf{y}|\mathbf{x})$ after a certain "burn-in" period, i.e., for $t > T_c$. Thus, samples for $t < T_c$ are discarded, whereas samples for $t > T_c$ are used to perform the desired inference task. Two problems associated to MCMC schemes are the difficulty in determining exactly when the chain has converged and the correlation among the generated samples (unlike the schemes in Section B.1, which produce i.i.d. samples).